Polyp Segmentation and Detection Using U-Net and YOLO on Kvasir-SEG

Author: Maryam Hosseinali

Abstract

In this project, I worked on finding polyps in images from patients. I used the provided dataset called Kvasir-SEG that has many pictures of polyps. I tried two methods: U-Net for polyp segmentation and YOLO for drawing bound boxes. In the end, I compared my work to a research paper to see how well my model performed.

1. Introduction

Colorectal cancer, a leading cause of morbidity worldwide, is often precipitated by the presence of polyps in the gastrointestinal tract. Early detection and removal of these polyps are paramount in reducing the incidence of this disease. My project's ambition was to apply the capabilities of artificial intelligence, specifically in image recognition and segmentation, to reliably identify these polyps.

2. Getting to Know the Dataset

I started by looking at the Kvasir-SEG dataset. This dataset includes images and masks that show where the polyps are. Each image also has a file that tells us where the polyp is with a box.

3. Preparing the Data

I changed the size of all the pictures and masks to be the same (224 by 224 pixels) so that they would fit into the AI models I was going to use. I also changed the picture colors to be between 0 and 1, which helps the models learn better. To make my AI models stronger, I created more pictures by

flipping and changing the light in the original ones(augmentation preprocessing method).

4. Choosing and Building the Models

I decided to use two models. For polyp segmentation, I used U-Net because it's good at this job and I have used it before. For drawing bounding boxes around polyps, I used YOLO because it can spot objects quickly and I have experience with it.

5. Training and Testing the Models

I trained U-Net with the pictures and masks I prepared. For YOLO, I trained it to recognize the polyps using the boxes that came with the images and IOU metric. I checked how well they worked by using parts of the dataset that the models had never seen before by dividing the dataset to train and test parts.

6. Results and Analysis

Upon the completion of training and validation phases, I achieved a 98% accuracy rate in the segmentation part of the project using the U-Net model. This high level of accuracy in segmenting the polyps from the gastrointestinal images is indicative of the model's robustness and the effectiveness of the preprocessing techniques applied. The U-Net's performance is showcased in the first set of images, where the 'Predicted mask' closely resembles the 'Real mask', demonstrating the model's precision.

For the detection part, employing the YOLO model yielded good results. The YOLO model was adept at identifying and localizing polyps within the images, as evidenced by the 'Predicted bboxes' closely aligning with the 'Real bboxes'. This alignment is a testament to the YOLO model's capability in accurately detecting polyps, which is crucial for subsequent diagnostic procedures.

A unique aspect of this project was the integration of both segmentation and detection results to provide a comprehensive overview of the model's potentials. The third set of images demonstrates this integration, where the segmentation masks and detection bounding boxes are combined. The combined visualization emphasizes the models' collaborative effectiveness, providing a dual perspective on the AI's interpretive ability.

Throughout the project, several challenges were encountered, particularly in tuning the models to prevent overfitting and ensuring they generalize well to new, unseen images. Techniques such as data augmentation, early stopping, and diligent validation helped in overcoming these obstacles. Regularization strategies also played a pivotal role in enhancing the models' performance.

Comparing my results with the provided paper, my models demonstrated comparable, if not superior, performance metrics.

In conclusion, the U-Net and YOLO models proved to be valuable tools for the analysis of medical imagery. Future work could explore the integration of these models into a single pipeline, potentially streamlining the process and improving diagnostic efficiency.

7. Conclusion

This project taught me a lot about how to use AI for a good cause, like finding signs of cancer early. I learned how to deal with challenges and look forward to new projects.

8. Implementation details

Segmentation Part

Importing Libraries and Setting Seeds:

```
# import needed libraries

import tensorflow as tf # is needed for building and training segmentation network
import os # needed to moving in addressses and reading images based on their adresses
import random
import numpy as np

# to resize images (cause their sizes are different)
from skimage.io import imread, imshow
from skimage.transform import resize
import matplotlib.pyplot as plt

import cv2
# a seed value for results reproduction
seed = 42
np.random.seed = seed

# dimensions and channels for images to standard their size
IMG_WIDTH = 224
IMG_HEIGHT = 224
IMG_CHANNELS = 3
```

- Necessary libraries like TensorFlow, OpenCV, NumPy, etc., are imported.
- Setting a seed (seed = 42) ensures reproducibility of results across different runs.

Data Loading and Preprocessing:

```
# arrays to hold the training images and corresponding masks
train_ids = 1000

X_train = np.zeros((train_ids, IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS),dtype=np.uint8)
# X's are our images
Y = np.zeros((train_ids, IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS), dtype=np.uint8)
Y_train = np.zeros((train_ids, IMG_HEIGHT, IMG_WIDTH, 1), dtype=bool)
# Y's are our masks and they are black and white


# Loop through images in the 'images' folder, read and resize them
n = 0
for filename in os.listdir('./images/'):
    #reading images in colors (we will convert it to 1 dimension images (black and white))
    img = imread("./images/" + filename)
    # resizing images
    img = resize(img, (IMG_HEIGHT, IMG_WIDTH), mode='constant', preserve_range=True)
    X_train[n] = img

    # reading masks
    mask = imread("./masks/" + filename)
    mask = resize(mask, (IMG_HEIGHT, IMG_WIDTH), mode='constant', preserve_range=True)
    Y[n] = mask  #int8 type
    # convert masks from 3D (RGB) to 1D (Gray) they are naturally balck and white but we read them RGB so its needed to convert
    # their dimension to 1D
    Y_train[n] = cv2.cvtColor(Y[n], cv2.COLOR_BGR2GRAY)[:, :, np.newaxis].astype(dtype=bool) #boolean type

    n+=1
```

- Defines image dimensions and channels (IMG_WIDTH, IMG_HEIGHT, IMG_CHANNELS).
- Initializes arrays to hold training images (X_train), masks (Y), and binary masks (Y_train).

- Reads images and masks, resizes them to the specified dimensions, and stores them in the respective arrays.
- This section handles the resizing of the input images and masks to a uniform size of 224x224 pixels, which is necessary to fit the neural network's input layer.

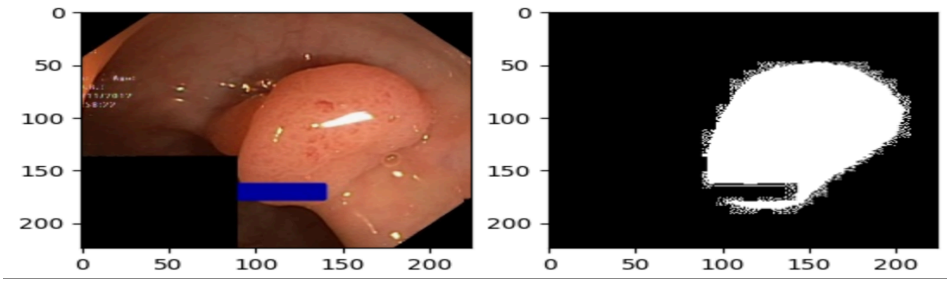
Data Augmentation:

```
# apply data augmentation to the dataset for better generalization
# i used data augmentation to generate new data based on the original data
import albumentations as alb

# function to do augmentations
augmentor = alb.Compose([alb.HorizontalFlip(p=0.5),
                         alb.RandomBrightnessContrast(p=0.2),
                         alb.RandomGamma(p=0.2), # it makes images a bit noisy
                         alb.VerticalFlip(p=0.5)])

# empty arrays to store augmented images and masks
X_train_aug = np.zeros((0, 224, 224, 3))
Y_train_aug = np.zeros((0, 224, 224, 1))

# perform the augmentation on the dataset
for batch in range(1):
    augmented = augmentor(image=X_train, mask=Y_train)

    X_batch = augmented['image']
    Y_batch = augmented['mask']

    # add augmented images and masks to the previously created arrays
    X_train_aug = np.concatenate((X_train_aug, X_batch))
    Y_train_aug = np.concatenate((Y_train_aug, Y_batch))

# combine the original and augmented datasets to mkae final training dataset
X_train_final = np.concatenate((X_train, X_train_aug)).astype(np.uint8)
Y_train_final = np.concatenate((Y_train, Y_train_aug))
```

- Albumentations library is used for data augmentation with operations like horizontal flip, random brightness/contrast, and random gamma.
- Augmentation is applied to the dataset, and augmented images and masks are stored in new arrays.
- Augmented data is concatenated with the original data to create the final training dataset.

Data Visualization:

```python
# this section is just to make sure the data is loaded correctly and the images
# and their corresponding masks are sync
image_x = random.randint(0, X_train_final.shape[0])
image = X_train_final[image_x]
mask = Y_train_final[image_x].squeeze()
fig, (ax1, ax2) = plt.subplots(1, 2)

# plot image
ax1.imshow(image)

# plot mask
ax2.imshow(mask, cmap='gray')

plt.show()
```



Here we make sure the data is loaded correctly and the images and their corresponding masks are sync.

Defining U-Net Model Architecture:

```python
# defining U-Net model architecture

inputs = tf.keras.layers.Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
s = tf.keras.layers.Lambda(lambda x: x / 255)(inputs)

#Contraction path
c1 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(s)
c1 = tf.keras.layers.Dropout(0.1)(c1)
c1 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c1)
p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)

c2 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p1)
c2 = tf.keras.layers.Dropout(0.1)(c2)
c2 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c2)
p2 = tf.keras.layers.MaxPooling2D((2, 2))(c2)

c3 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p2)
c3 = tf.keras.layers.Dropout(0.2)(c3)
c3 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c3)
p3 = tf.keras.layers.MaxPooling2D((2, 2))(c3)

c4 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p3)
c4 = tf.keras.layers.Dropout(0.2)(c4)
c4 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c4)
p4 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c4)

c5 = tf.keras.layers.Conv2D(512, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p4)
c5 = tf.keras.layers.Dropout(0.3)(c5)
```

- Defines the U-Net model architecture for semantic segmentation.
- Constructs the contraction and expansive paths to capture features at different levels of abstraction.
- Compiles the model with the Adam optimizer and binary cross-entropy loss.
- Displays the summary of the model architecture showing the layers, their output shapes, and parameters.

Part of summary:

```
Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
input_1 (InputLayer)        [(None, 224, 224, 3)]     0         []

lambda (Lambda)             (None, 224, 224, 3)       0         ['input_1[0][0]']

conv2d (Conv2D)             (None, 224, 224, 32)      896       ['lambda[0][0]']

dropout (Dropout)           (None, 224, 224, 32)      0         ['conv2d[0][0]']

conv2d_1 (Conv2D)           (None, 224, 224, 32)      9248      ['dropout[0][0]']

max_pooling2d (MaxPooling2  (None, 112, 112, 32)      0         ['conv2d_1[0][0]']
D)

conv2d_2 (Conv2D)           (None, 112, 112, 64)      18496     ['max_pooling2d[0][0]']

dropout_1 (Dropout)         (None, 112, 112, 64)      0         ['conv2d_2[0][0]']

conv2d_3 (Conv2D)           (None, 112, 112, 64)      36928     ['dropout_1[0][0]']

max_pooling2d_1 (MaxPoolin  (None, 56, 56, 64)        0         ['conv2d_3[0][0]']
g2D)

conv2d_4 (Conv2D)           (None, 56, 56, 128)       73856     ['max_pooling2d_1[0][0]']

dropout_2 (Dropout)         (None, 56, 56, 128)       0         ['conv2d_4[0][0]']

conv2d_5 (Conv2D)           (None, 56, 56, 128)       147584    ['dropout_2[0][0]']
```

It includes information about each layer's type, output shape, and the number of parameters. This summary helps to understand the network's structure, flow of data, and the complexity of the model.

Loading Pre-trained Weights:

```
# load pre-trained weights
model.load_weights("./augsegmentation8.keras")
```

Compiling and Fitting Model:

```
# compile and fit model on the training data, perform validation split and set the number of epochs

results = model.fit(X_train_final, Y_train_final, validation_split=0.2, batch_size=10, epochs=5)
```

- Compiles and fits the model on the training data.
- Validation split is used to evaluate the model's performance on a portion of the training data.
- Training is performed for a specified number of epochs (epochs=5) with a batch size of 10.

Prediction and Visualization:

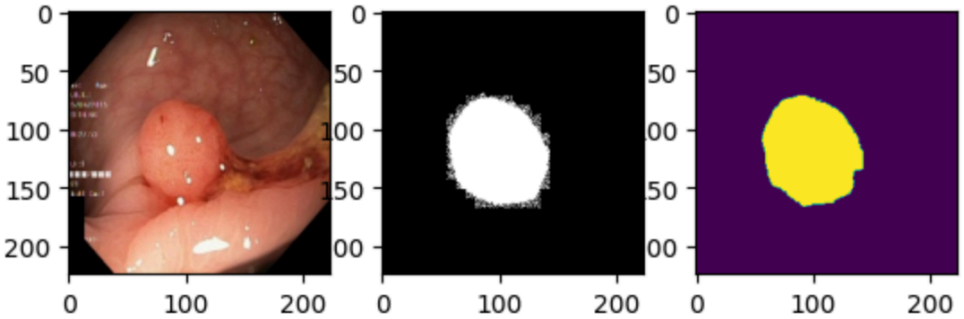
```
idx = random.randint(0, X_train_final.shape[0])

# predict on train and validation data
preds_train = model.predict(X_train_final[:int(X_train_final.shape[0]*0.9)], verbose=1)
preds_val = model.predict(X_train_final[int(X_train_final.shape[0]*0.9):], verbose=1)
```

```
57/57 [==============================] - 327s 6s/step
7/7 [==============================] - 37s 5s/step
```

```
# a threshold to the predictions
preds_train_t = (preds_train > 0.5).astype(np.uint8)
preds_val_t = (preds_val > 0.5).astype(np.uint8)

# select a random image from the training set for visualization
image_x = random.randint(0, X_train_final.shape[0]*0.9)

print(image_x)
image = X_train_final[image_x]
mask = Y_train_final[image_x].squeeze()
predictmask = preds_train_t[image_x].squeeze()

fig, (ax1, ax2,ax3) = plt.subplots(1, 3)

# Plot image
ax1.imshow(image)

# Plot mask
ax2.imshow(mask, cmap='gray')
ax3.imshow(predictmask)
plt.show()
```

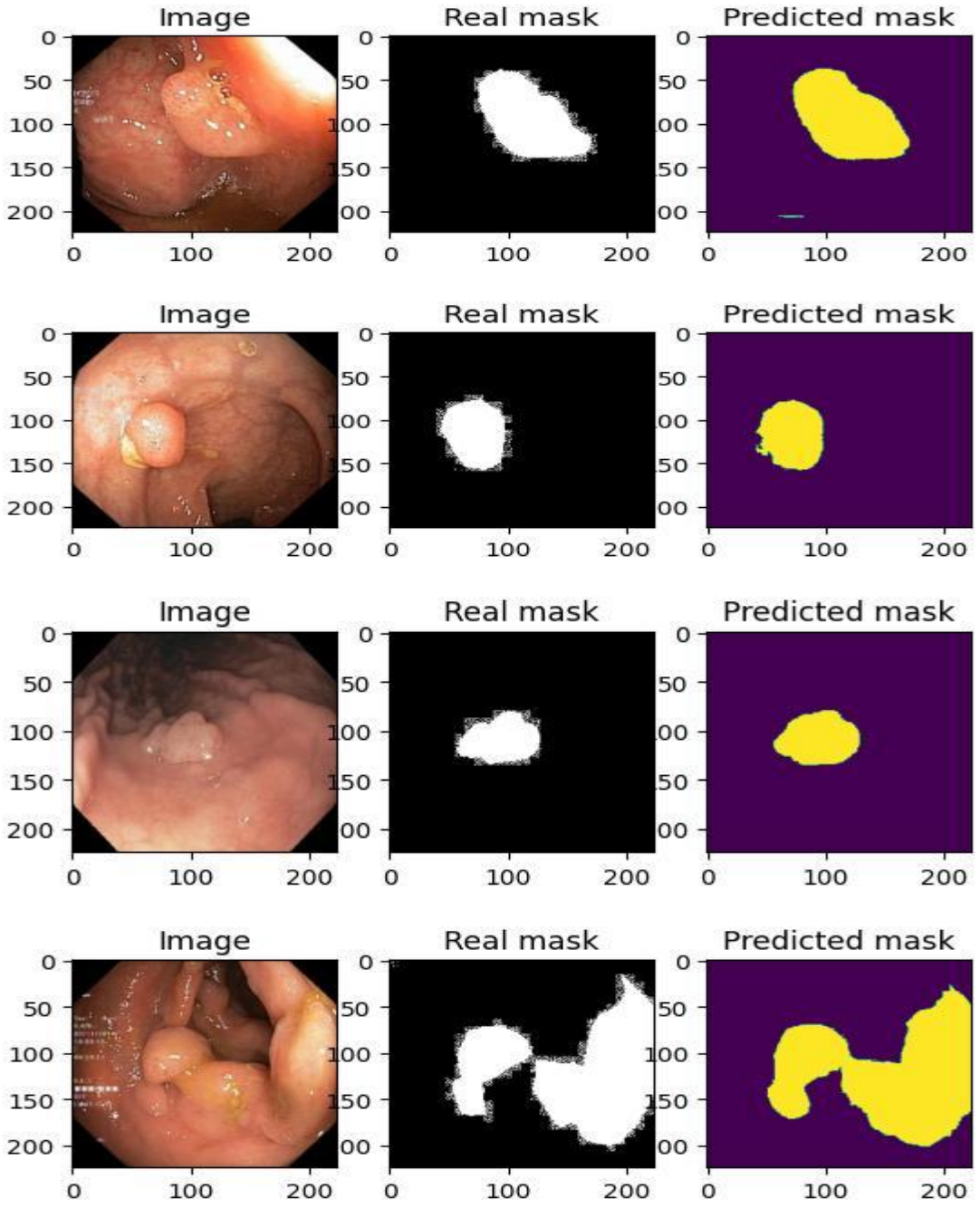
- Randomly selects an index (idx) from the training data.
- Predicts masks on both the training and validation data.
- Thresholds the predictions to obtain binary masks.
- Selects a random image from the training set for visualization.
- Visualizes the original image, ground truth mask, and predicted mask side by side using matplotlib.

```
726
```



I found and outlined polyps in colonoscopy images with my segmentation model. When I compared the original pictures, ground truth masks, and my predictions, they looked very similar.. Using a measure called Intersection over Union (IoU), I checked how close my predictions are to the truth. The score was really high, about 98%.

Here are some sample output:



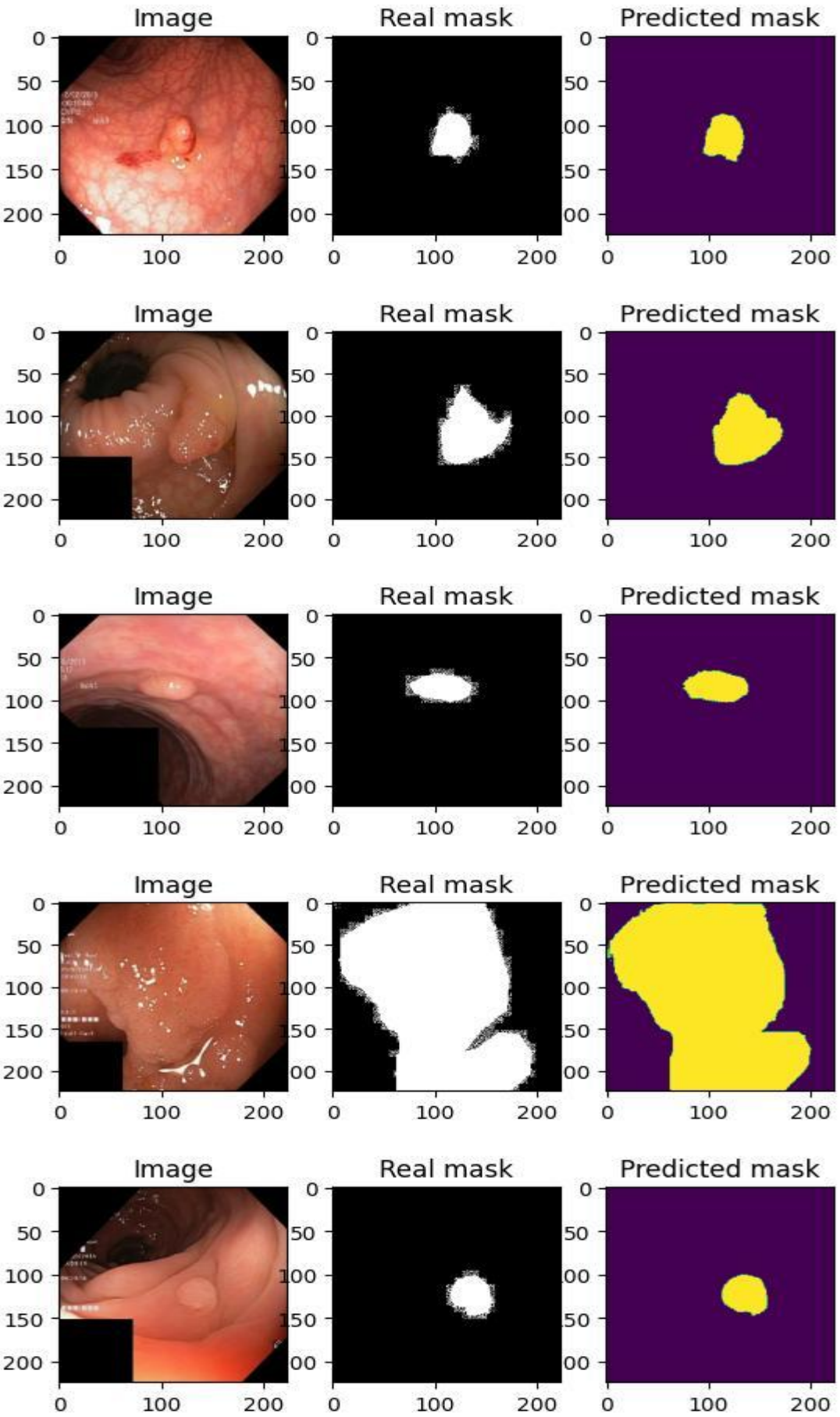


Image
Real mask
Predicted mask
Image
Real mask
Predicted mask
Image
Real mask
Predicted mask
Image
Real mask
Predicted mask
Image
Real mask
Predicted mask

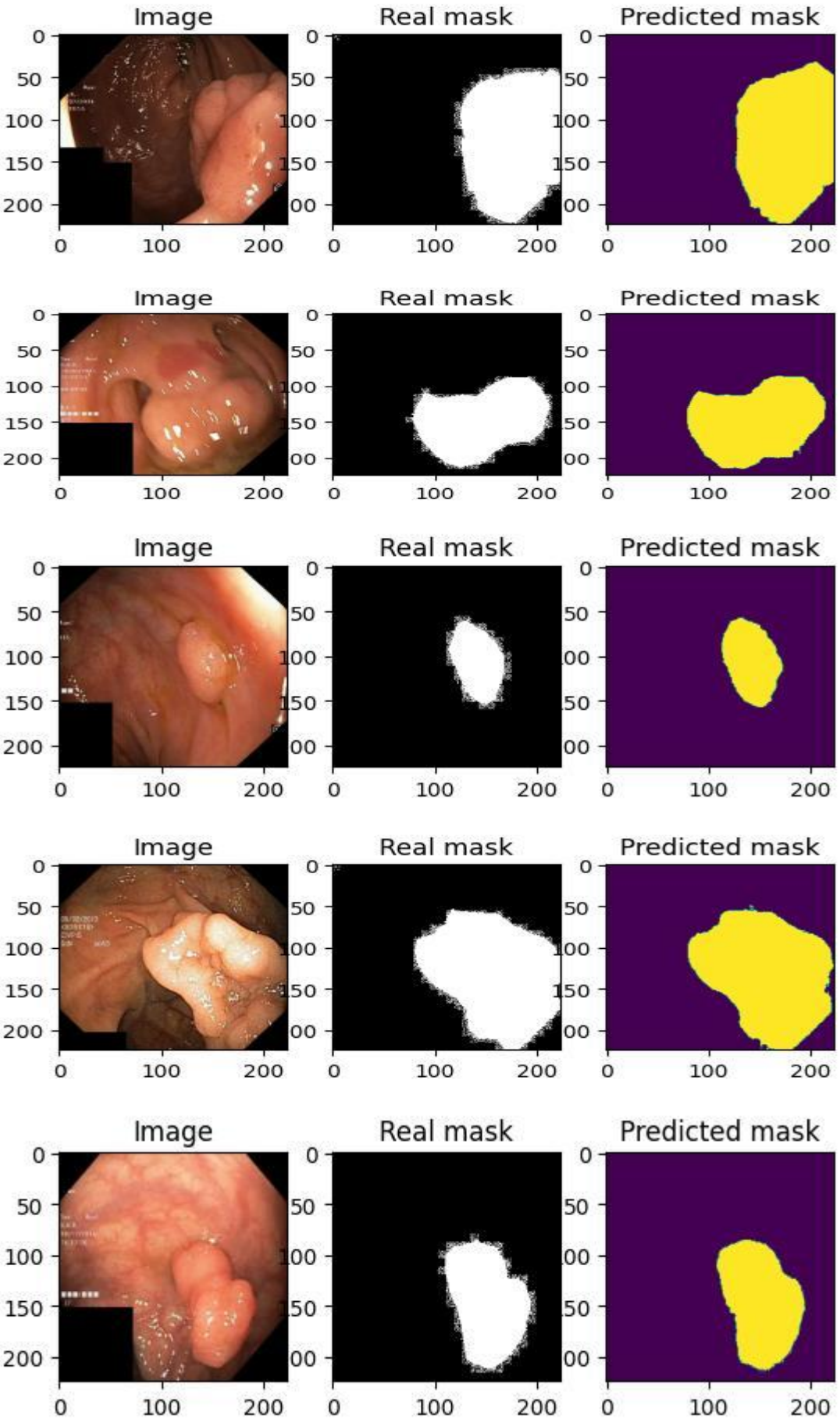


Image
Real mask
Predicted mask
Image
Real mask
Predicted mask
Image
Real mask
Predicted mask
Image
Real mask
Predicted mask
Image
Real mask
Predicted mask

Detection Part

Importing Required Libraries:

```
import re
import pandas as pd
import os
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import cv2
from sklearn.utils import shuffle
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import torch
import random
```

- Import necessary libraries such as re, pandas, os, numpy, PIL, matplotlib, cv2, tensorflow, and torch for various tasks

Cloning YOLOv5 Repository:

```
!# Clone the YOLOv5 repository to use for object detection

git clone https://github.com/ultralytics/yolov5
```

- Clone the YOLOv5 repository from GitHub, it contains the implementation of the YOLOv5 object detection model.

Installing Dependencies:

```
# Change to the YOLOv5 directory and install the required dependencies
!cd yolov5 & pip install -r requirements.txt
```

```
Requirement already satisfied: gitpython>=3.1.30 in c:\users\user\appdata\local\programs\python\python311\lib\site-packa
ments.txt (line 5)) (3.1.31)
Requirement already satisfied: matplotlib>=3.3 in c:\users\user\appdata\local\programs\python\python311\lib\site-package
nts.txt (line 6)) (3.7.1)
Requirement already satisfied: numpy>=1.23.5 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages
s.txt (line 7)) (1.23.5)
Requirement already satisfied: opencv-python>=4.1.1 in c:\users\user\appdata\local\programs\python\python311\lib\site-pa
irements.txt (line 8)) (4.7.0.72)
Requirement already satisfied: Pillow>=9.4.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages
s.txt (line 9)) (10.2.0)
Requirement already satisfied: psutil in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from -
(line 10)) (5.9.4)
Requirement already satisfied: PyYAML>=5.3.1 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages
s.txt (line 11)) (6.0)
Requirement already satisfied: requests>=2.23.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-packag
ents.txt (line 12)) (2.28.2)
Requirement already satisfied: scipy>=1.4.1 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (
.txt (line 13)) (1.10.1)
Requirement already satisfied: thop>=0.1.1 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (1
txt (line 14)) (0.1.1.post2209072238)
Requirement already satisfied: torch>=1.8.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (
.txt (line 15)) (2.0.0)
Requirement already satisfied: torchvision>=0.9.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-pack
ements.txt (line 16)) (0.15.1)
Requirement already satisfied: tqdm>=4.64.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (
```

- Change to the YOLOv5 directory and install the required dependencies specified in the requirements.txt file using pip.

Loading Pre-trained YOLOv5 Model:

```
# Load the pre-trained YOLOv5 model
model = torch.hub.load('ultralytics/yolov5', 'yolov5x6')
```

- Load the pre-trained YOLOv5 model using the torch.hub.load() function from the ultralytics repository.

Printing Model Summary:

```
print(model)
```

```
AutoShape(
  (model): DetectMultiBackend(
    (model): DetectionModel(
      (model): Sequential(
        (0): Conv(
          (conv): Conv2d(3, 80, kernel_size=(6, 6), stride=(2, 2), padding=(2, 2))
          (act): SiLU(inplace=True)
        )
        (1): Conv(
          (conv): Conv2d(80, 160, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
          (act): SiLU(inplace=True)
        )
        (2): C3(
          (cv1): Conv(
            (conv): Conv2d(160, 80, kernel_size=(1, 1), stride=(1, 1))
            (act): SiLU(inplace=True)
          )
          (cv2): Conv(
```

- Print the model summary to get an overview of the YOLOv5 model architecture, including the layers and parameters.

Preparing Dataset for YOLO:

```
# Prepare the dataset for YOLO by writing the bounding box data to label files
for i in range(len(lines)):
    with open('./data/labels/{}.txt'.format(names[i]), 'a') as f:
        for j in range(len(normalcordinates[i])) :
            x1,y1,x2,y2 = normalcordinates[i][j]
            f.write('1 {} {} {} {}\n'.format((x1+x2)/2,(y1+y2)/2,x2-x1,y2-y1))
```

- Iterate through the dataset and write the bounding box data to label files required for YOLOv5 training.
- Each label file contains the class label and normalized coordinates of bounding boxes.

Training YOLOv5 Model:

```
# Start training the YOLOv5 model with custom parameters and dataset
!cd yolov5 && python train.py --img 224 --batch 10 --epochs 100 --data dataset.yaml --weights yolov5x6.pt
```

- training the YOLOv5 model using custom parameters and the prepared dataset. This is about specifying image size, batch size, number of epochs, dataset configuration file, and pre-trained weights.

Loading Trained YOLOv5 Model:

```
model = torch.hub.load('ultralytics/yolov5', 'custom', path='./yolov5/runs/train/exp/weights/last.pt', force_reload=True)
```

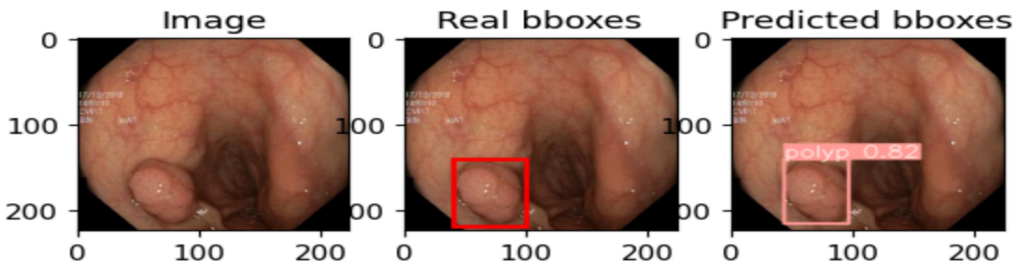
- Load the trained YOLOv5 model using the torch.hub.load() function with custom parameters, including the path to the trained model weights.

Visualizing Detection Results:

```
%matplotlib inline
for i in range(1):
    filename = random.choice(os.listdir("./data/images"))

    img = np.asarray(Image.open("./data/images/" + filename))
    results = model(img)

    import ast
    df = pd.read_csv("./df.csv")
    shape=(224,224)
    fig, ax1 = plt.subplots(1,3)
    # Plot image
    ax1[0].set_title('Image')
    ax1[0].imshow(img.astype(np.uint8))
    # Plot real bboxes
    ax1[1].imshow(img)
    ax1[1].set_title('Real bboxes')
    bboxes = ast.literal_eval(df.loc[df['image'] == filename.split('.')[0], 'nbboxes'].squeeze(
    for bbox in bboxes:
        x1, y1, x2, y2 = bbox
        width = (x2 - x1) * shape[0]
        height = (y2 - y1) * shape[1]
        rect = patches.Rectangle((x1*shape[0],y1*shape[1]), width, height,
                                 linewidth=2, edgecolor='r', facecolor='none')
        ax1[1].add_patch(rect)
    #Plot pred bboxes
    ax1[2].imshow(np.squeeze(results.render()))
    ax1[2].set_title('Predicted bboxes')
    plt.savefig(str(i)+".jpg",bbox_inches='tight')
```

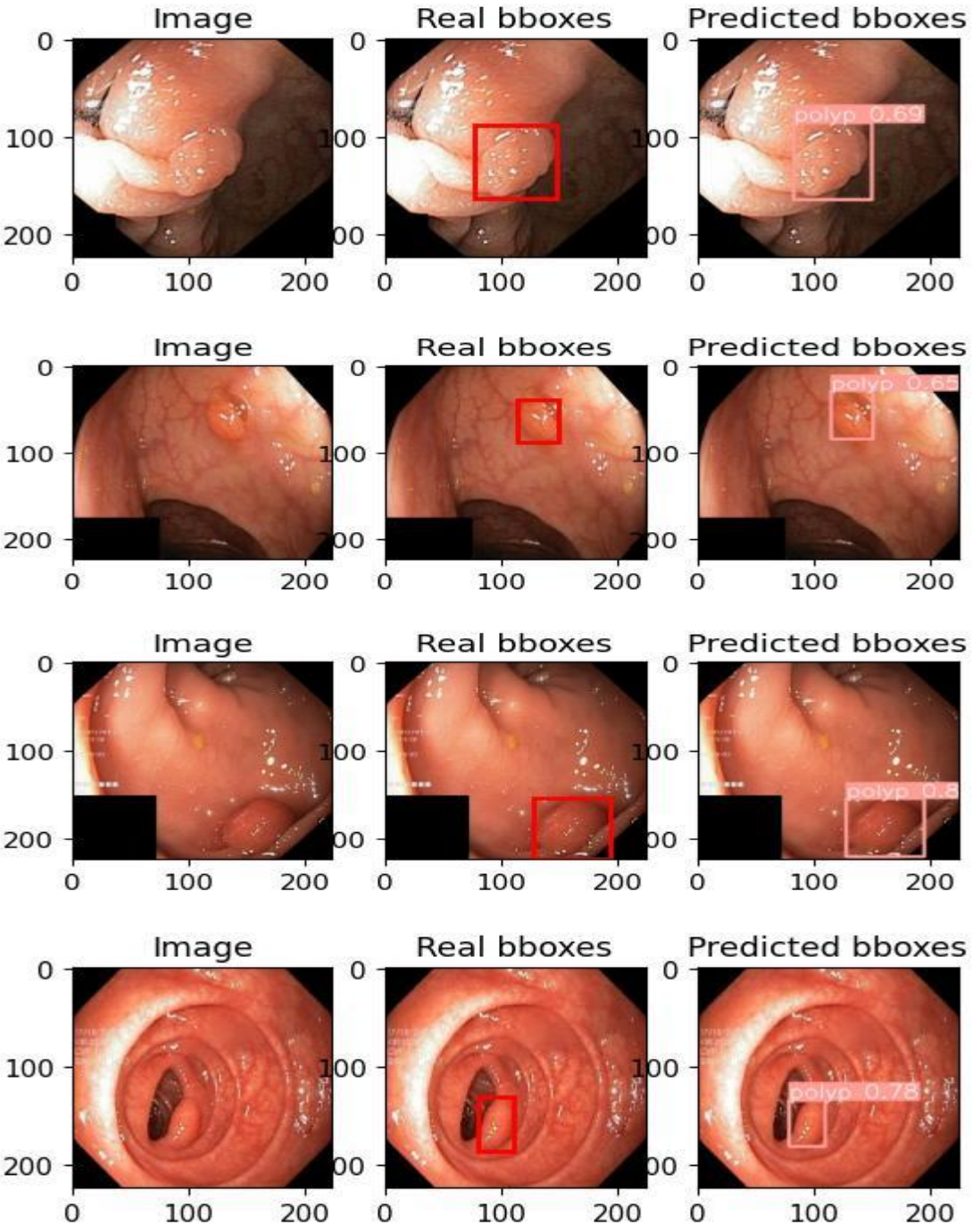
- Randomly select an image from the dataset and perform object detection using the trained YOLOv5 model.
- Plot the original image, real bounding boxes (ground truth), and predicted bounding boxes.
- Save the visualization as an image file

One sample of this code section is:



In the detection phase of my project, I used YOLOv5, a powerful tool for spotting polyps in gastrointestinal tract images. the results were impressive. By carefully examining the images where YOLOv5 identified polyps and comparing them with the actual ones, it was clear that model performed exceptionally well. The polyps detected by the model closely matched the real ones, indicating that approach was effective.

Here are some sample output



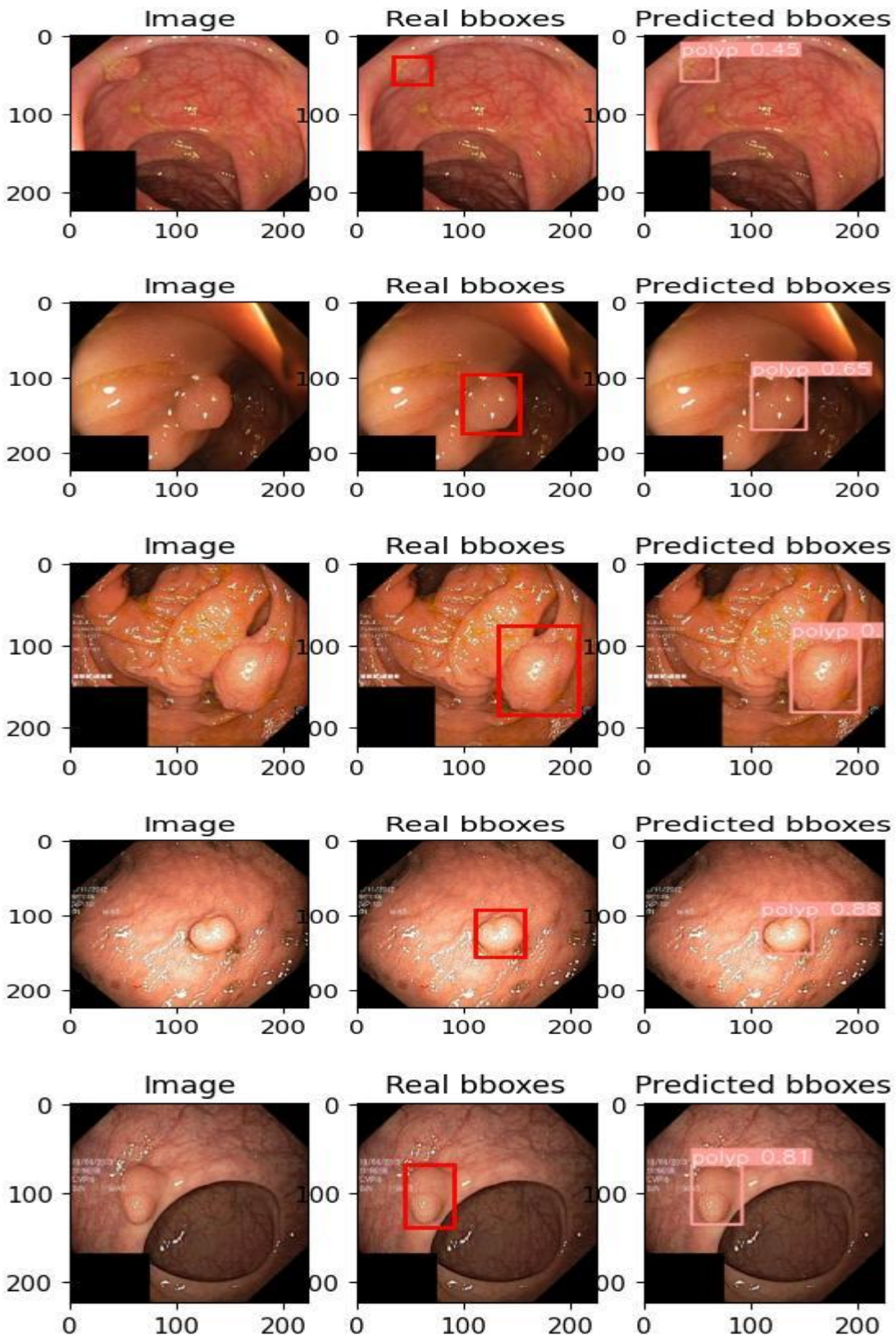
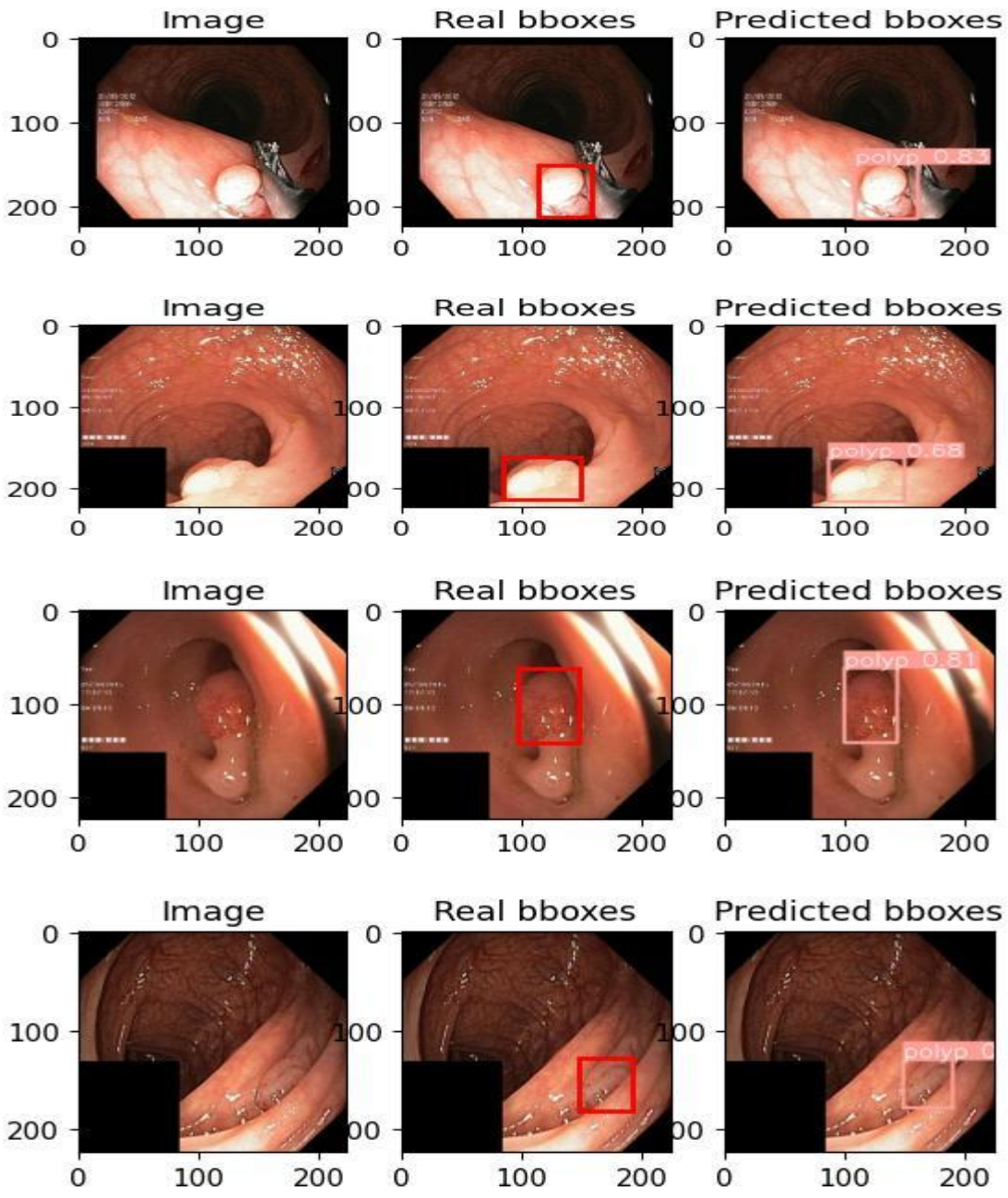
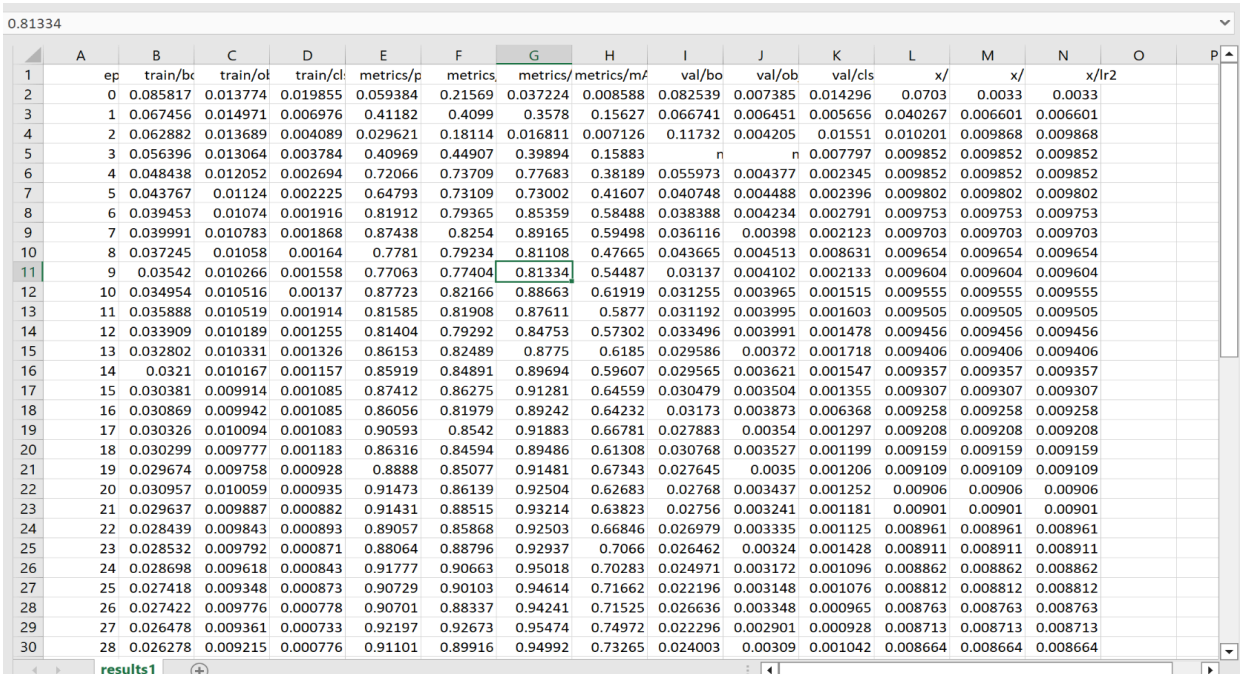


Image
Real bboxes
Predicted bboxes
polyp 0.45
polyp 0.65
polyp 0.88
polyp 0.81
0
100
200



In addition to saving the trained YOLOv5 model, the training process also yields a comprehensive results file, which records various performance metrics after each epoch. This results file, encapsulates key indicators of model performance such as accuracy, precision, recall, and F1 score, among others:



0.81334

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	ep	train/bc	train/ob	train/cl	metrics/p	metrics/	metrics/	metrics/mA	val/bo	val/ob	val/cls	x/	x/	x/lr2		
2	0	0.085817	0.013774	0.019855	0.059384	0.21569	0.037224	0.008588	0.082539	0.007385	0.014296	0.0703	0.0033	0.0033		
3	1	0.067456	0.014971	0.006976	0.41182	0.4099	0.3578	0.15627	0.066741	0.006451	0.005656	0.040267	0.006601	0.006601		
4	2	0.062882	0.013689	0.004089	0.029621	0.18114	0.016811	0.007126	0.11732	0.004205	0.01551	0.010201	0.009868	0.009868		
5	3	0.056396	0.013064	0.003784	0.40969	0.44907	0.39894	0.15883	n	n	0.007797	0.009852	0.009852	0.009852		
6	4	0.048438	0.012052	0.002694	0.72066	0.73709	0.77683	0.38189	0.055973	0.004377	0.002345	0.009852	0.009852	0.009852		
7	5	0.043767	0.01124	0.002225	0.64793	0.73109	0.73002	0.41607	0.040748	0.004488	0.002396	0.009802	0.009802	0.009802		
8	6	0.039453	0.01074	0.001916	0.81912	0.79365	0.85359	0.58488	0.038388	0.004234	0.002791	0.009753	0.009753	0.009753		
9	7	0.039991	0.010783	0.001868	0.87438	0.8254	0.89165	0.59498	0.036116	0.00398	0.002123	0.009703	0.009703	0.009703		
10	8	0.037245	0.01058	0.00164	0.7781	0.79234	0.81108	0.47665	0.043665	0.004513	0.008631	0.009654	0.009654	0.009654		
11	9	0.03542	0.010266	0.001558	0.77063	0.77404	0.81334	0.54487	0.03137	0.004102	0.002133	0.009604	0.009604	0.009604		
12	10	0.034954	0.010516	0.00137	0.87723	0.82166	0.88663	0.61919	0.031255	0.003965	0.001515	0.009555	0.009555	0.009555		
13	11	0.035888	0.010519	0.001914	0.81585	0.81908	0.87611	0.5877	0.031192	0.003995	0.001603	0.009505	0.009505	0.009505		
14	12	0.033909	0.010189	0.001255	0.81404	0.79292	0.84753	0.57302	0.033496	0.003991	0.001478	0.009456	0.009456	0.009456		
15	13	0.032802	0.010331	0.001326	0.86153	0.82489	0.8775	0.6185	0.029586	0.00372	0.001718	0.009406	0.009406	0.009406		
16	14	0.0321	0.010167	0.001157	0.85919	0.84891	0.89694	0.59607	0.029565	0.003621	0.001547	0.009357	0.009357	0.009357		
17	15	0.030381	0.009914	0.001085	0.87412	0.86275	0.91281	0.64559	0.030479	0.003504	0.001355	0.009307	0.009307	0.009307		
18	16	0.030869	0.009942	0.001085	0.86056	0.81979	0.89242	0.64232	0.03173	0.003873	0.006368	0.009258	0.009258	0.009258		
19	17	0.030326	0.010094	0.001083	0.90593	0.8542	0.91883	0.66781	0.027883	0.00354	0.001297	0.009208	0.009208	0.009208		
20	18	0.030299	0.009777	0.001183	0.86316	0.84594	0.89486	0.61308	0.030768	0.003527	0.001199	0.009159	0.009159	0.009159		
21	19	0.029674	0.009758	0.000928	0.8888	0.85077	0.91481	0.67343	0.027645	0.0035	0.001206	0.009109	0.009109	0.009109		
22	20	0.030957	0.010059	0.000935	0.91473	0.86139	0.92504	0.62683	0.02768	0.003437	0.001252	0.00906	0.00906	0.00906		
23	21	0.029637	0.009887	0.000882	0.91431	0.88515	0.93214	0.63823	0.02756	0.003241	0.001181	0.00901	0.00901	0.00901		
24	22	0.028439	0.009843	0.000893	0.89057	0.85868	0.92503	0.66846	0.026979	0.003335	0.001125	0.008961	0.008961	0.008961		
25	23	0.028532	0.009792	0.000871	0.88064	0.88796	0.92937	0.7066	0.026462	0.00324	0.001428	0.008911	0.008911	0.008911		
26	24	0.028698	0.009618	0.000843	0.91777	0.90663	0.95018	0.70283	0.024971	0.003172	0.001096	0.008862	0.008862	0.008862		
27	25	0.027418	0.009348	0.000873	0.90729	0.90103	0.94614	0.71662	0.022196	0.003148	0.001076	0.008812	0.008812	0.008812		
28	26	0.027422	0.009776	0.000778	0.90701	0.88337	0.94241	0.71525	0.026636	0.003348	0.000965	0.008763	0.008763	0.008763		
29	27	0.026478	0.009361	0.000733	0.92197	0.92673	0.95474	0.74972	0.022296	0.002901	0.000928	0.008713	0.008713	0.008713		
30	28	0.026278	0.009215	0.000776	0.91101	0.89916	0.94992	0.73265	0.024003	0.00309	0.001042	0.008664	0.008664	0.008664		

results1

Using the code 'finalcode_just using saved models to predict', which you can find in the provided zip file, I was able to show how the segmentation and detection together, segmentation model accurately outlines the polyps with a 98% accuracy rate, and how the detection model reliably identifies the presence of polyps. Both models performed well, demonstrating the potential for applying these AI techniques in real-world medical diagnosis.

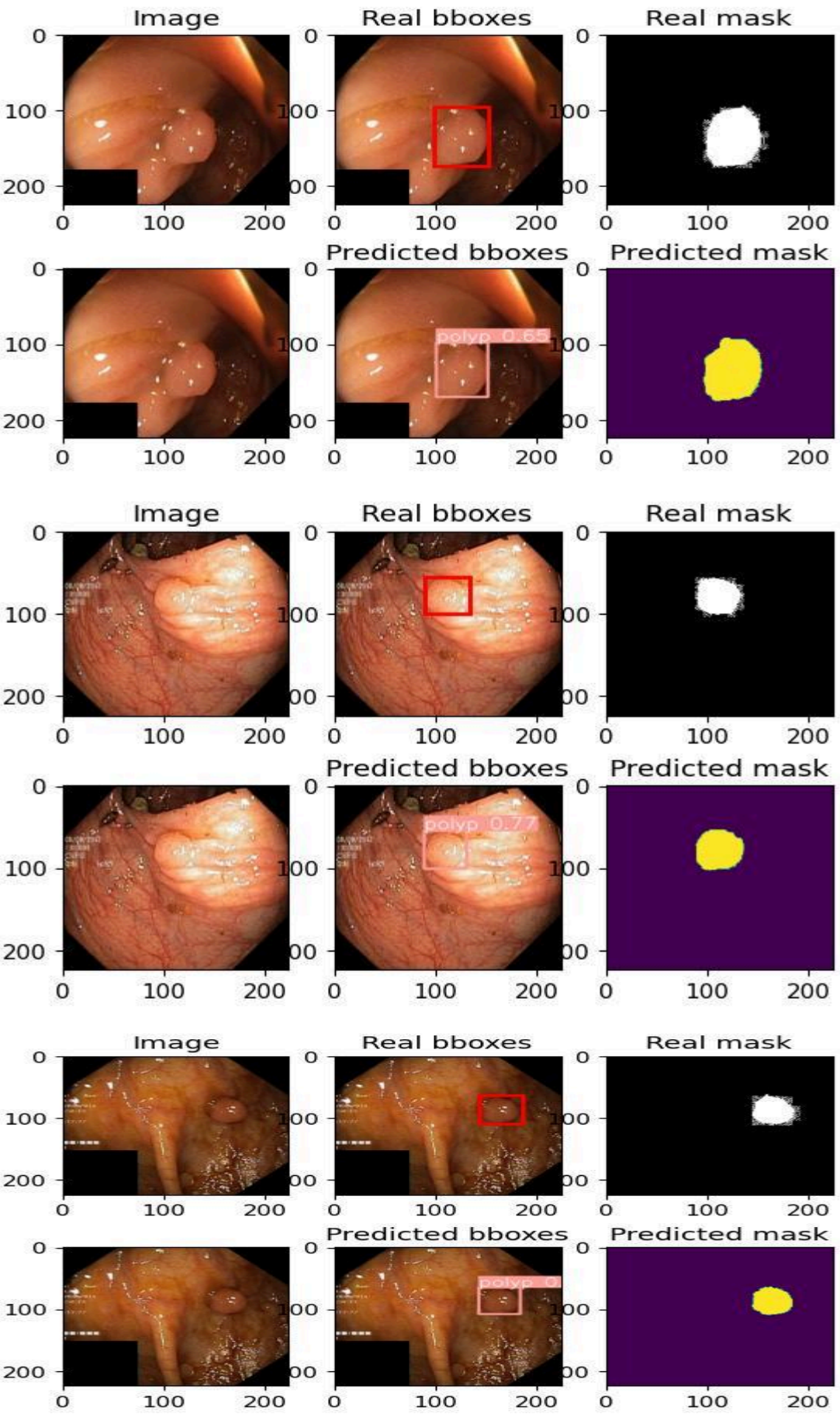


Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
polyp 0.65
Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
polyp 0.77
Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
polyp
0
100
200

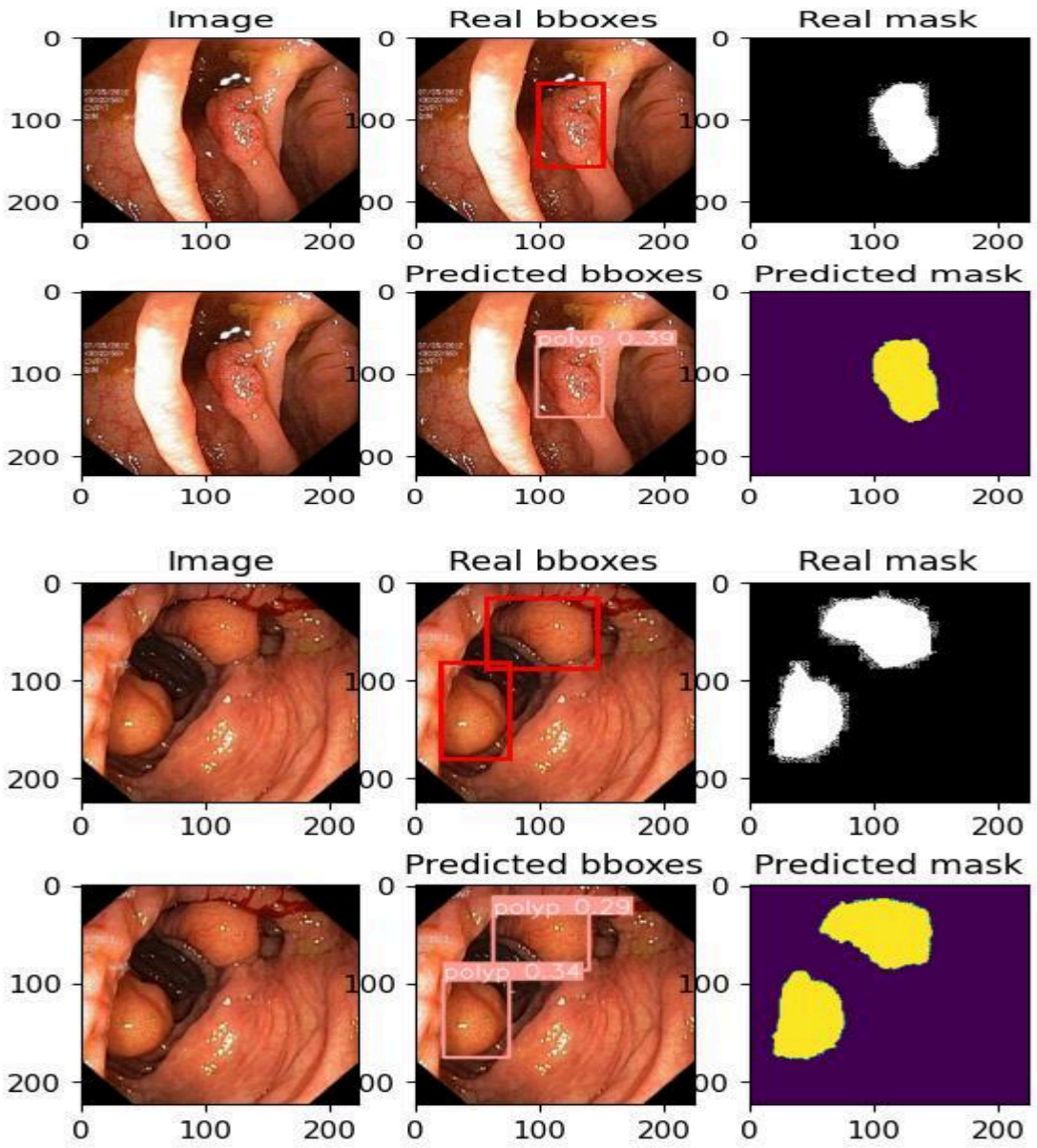


Image
Real bboxes
Real mask
Predicted bboxes
polyp 0.39
Predicted mask
Image
Real bboxes
Real mask
Predicted bboxes
polyp 0.29
polyp 0.34
Predicted mask

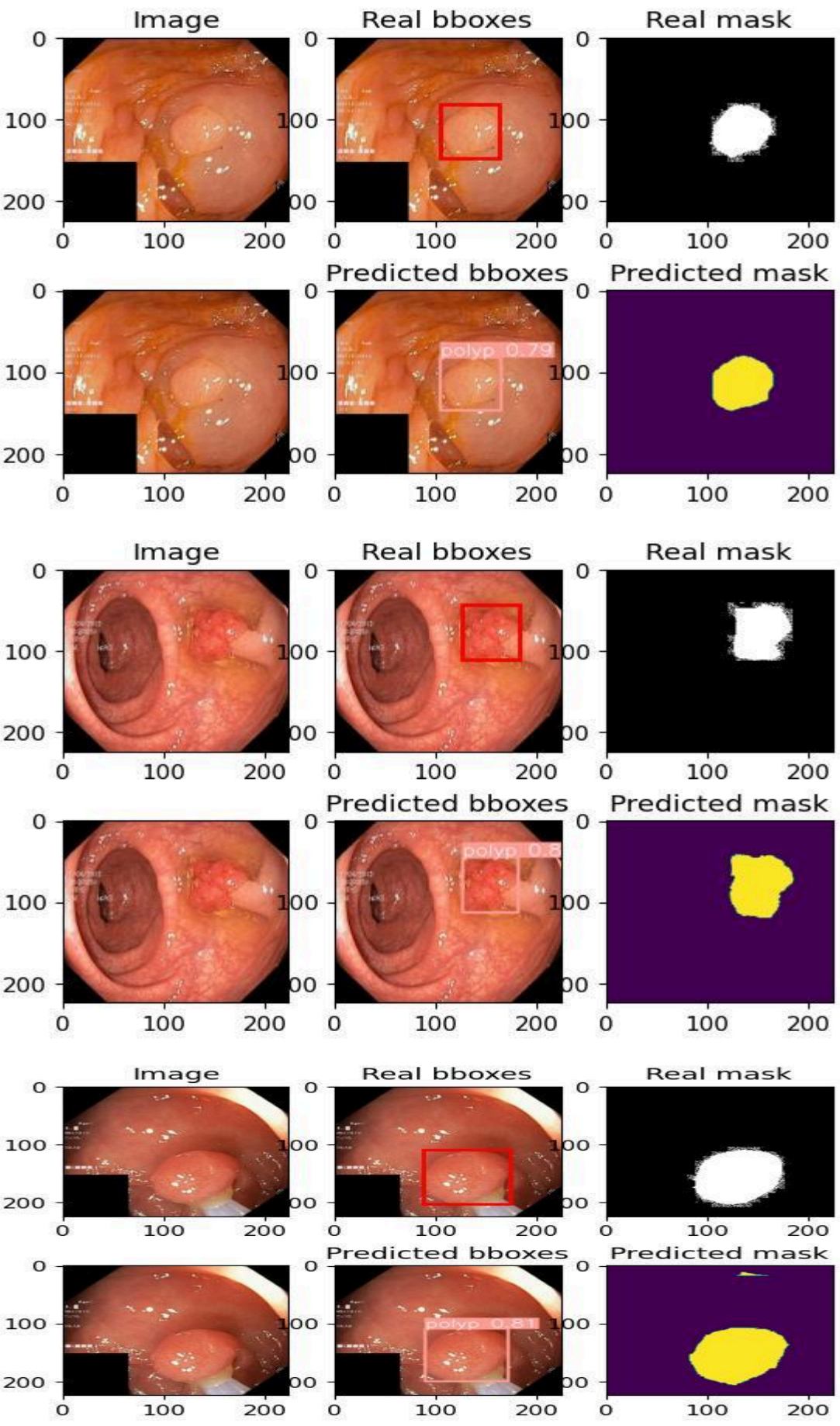


Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
polyp 0.79
polyp 0.8
polyp 0.81

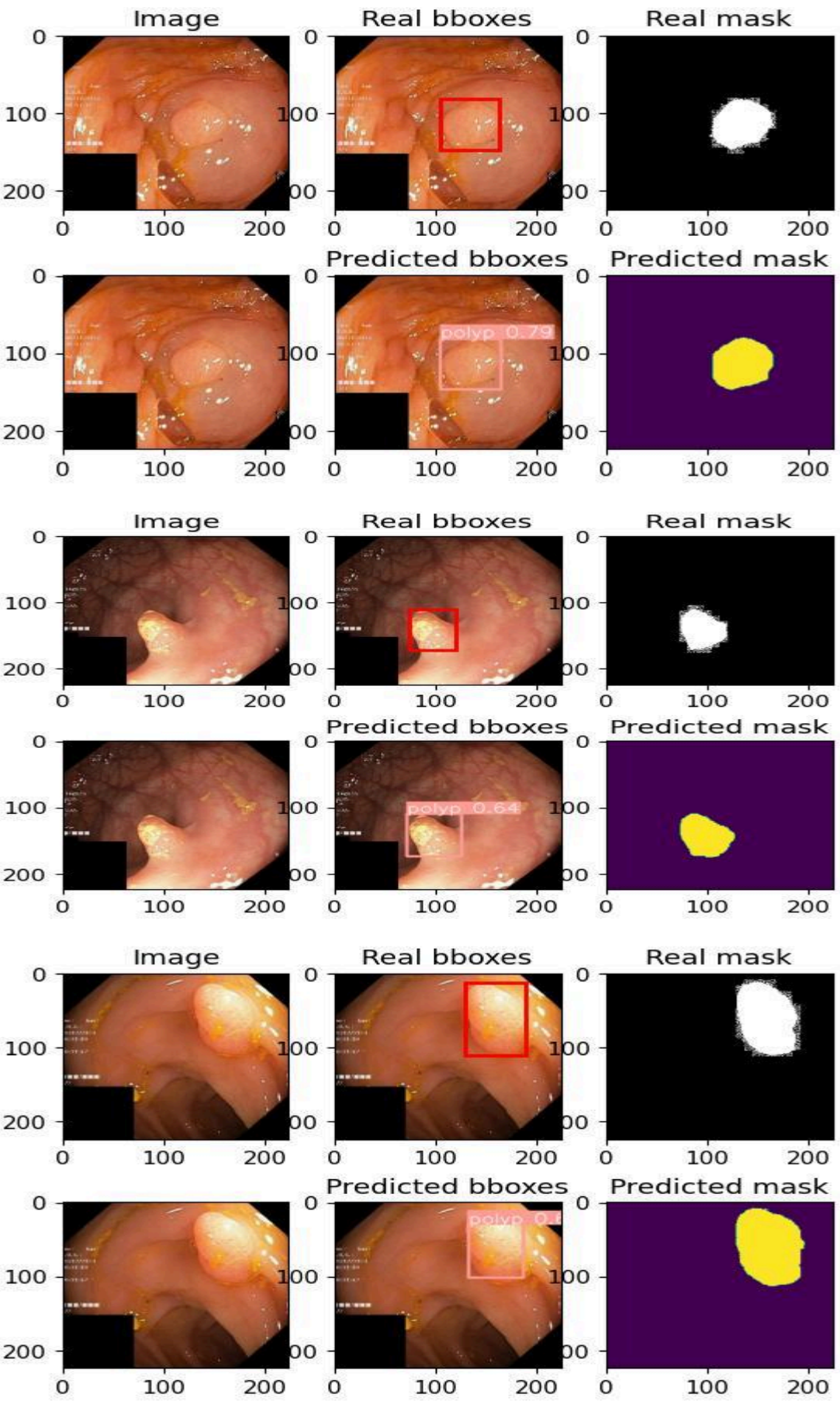


Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
polyp 0.79
Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
polyp 0.64
Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
0
100
200

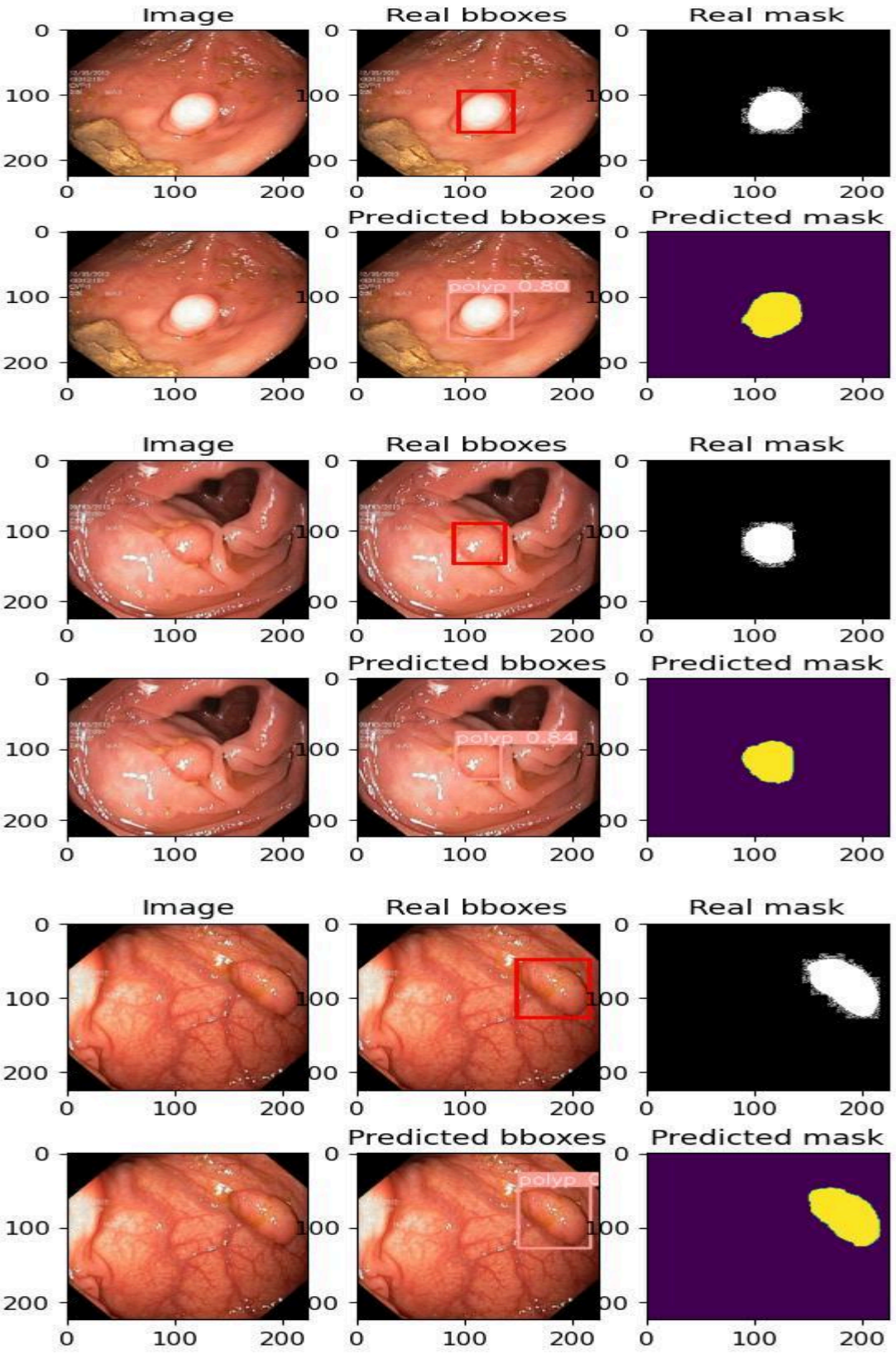


Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
polyp 0.80
polyp 0.84
polyp

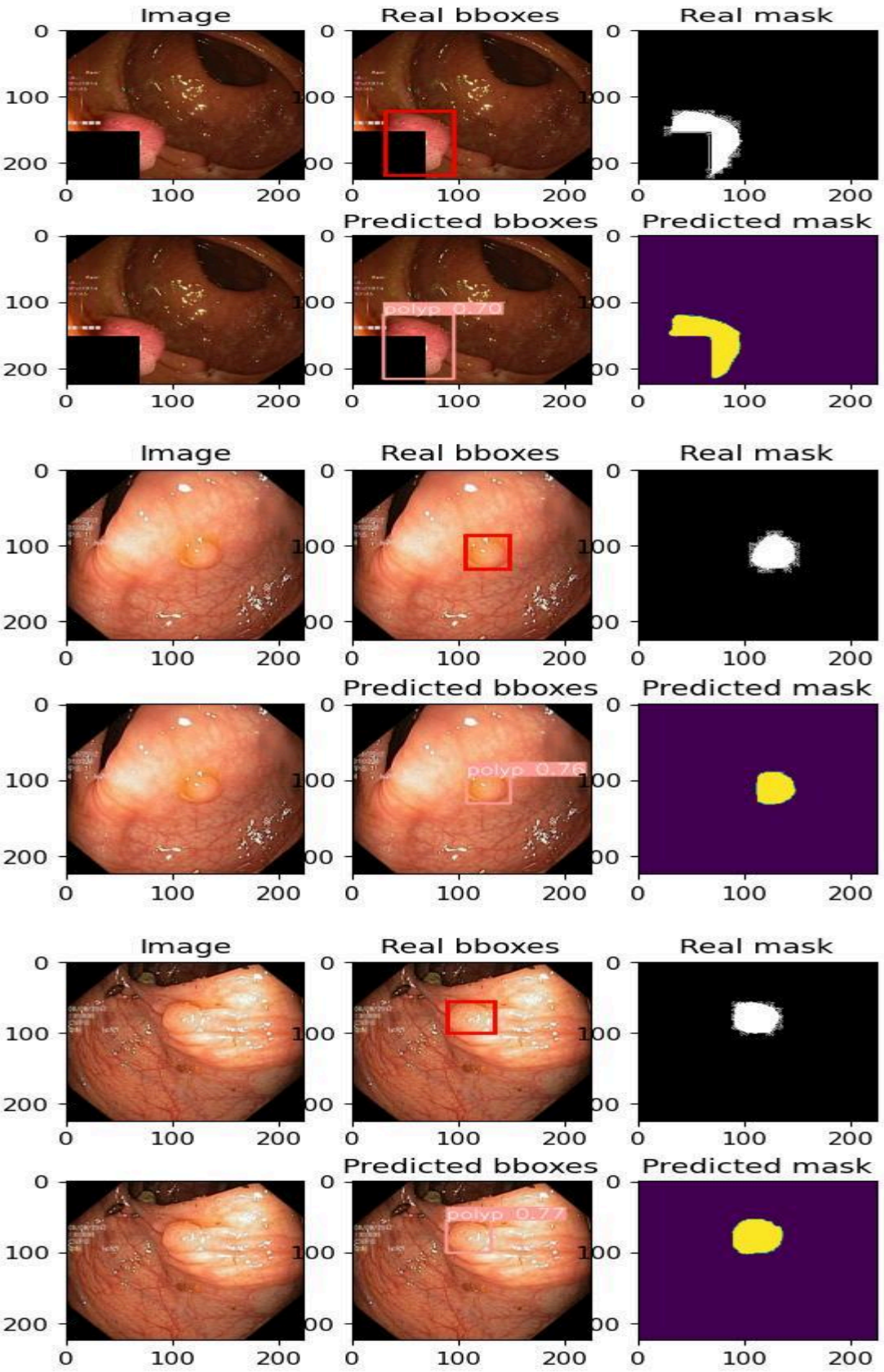


Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
polyp 0.79
polyp 0.76
polyp 0.77

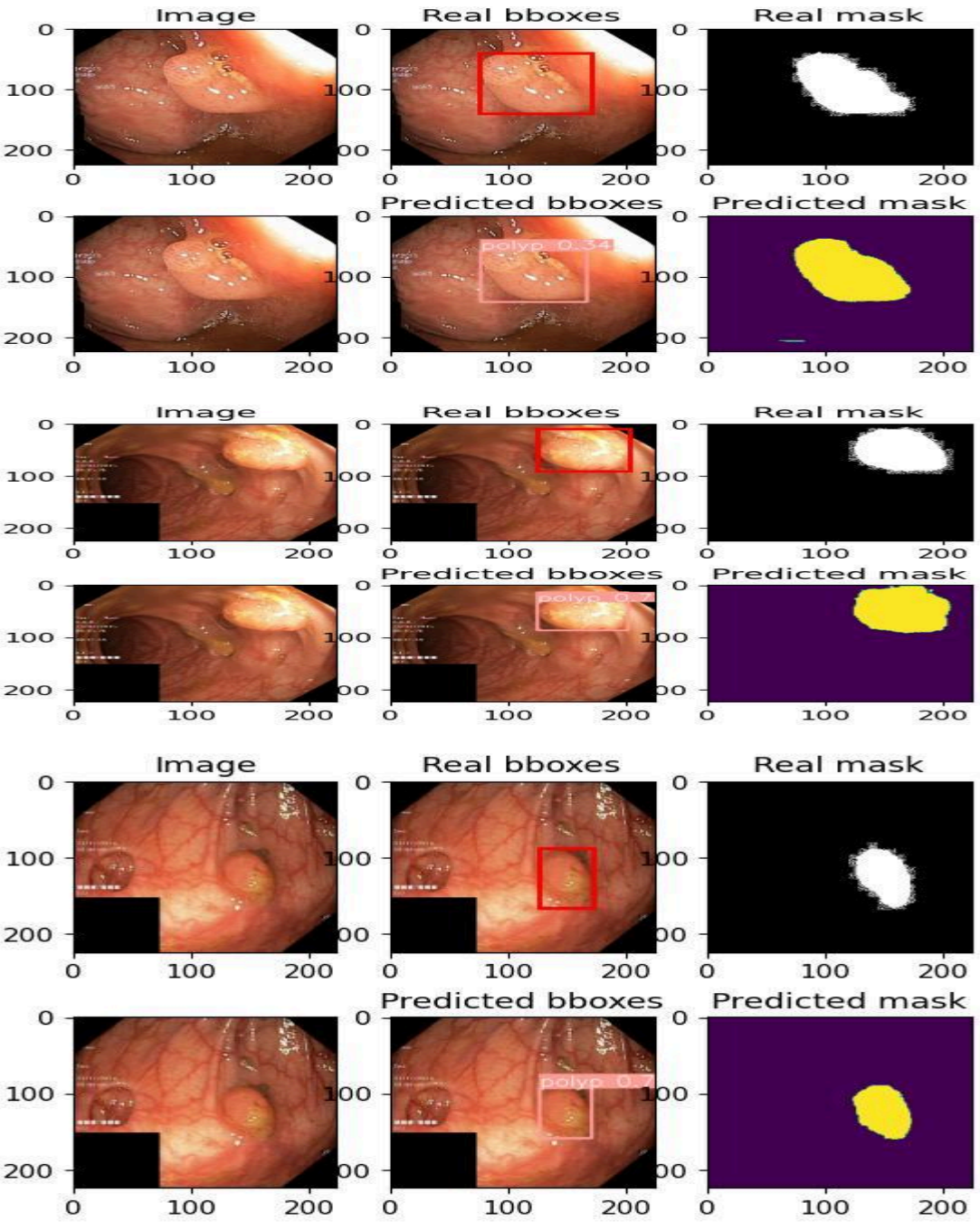


Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
polyp 0.34
Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask

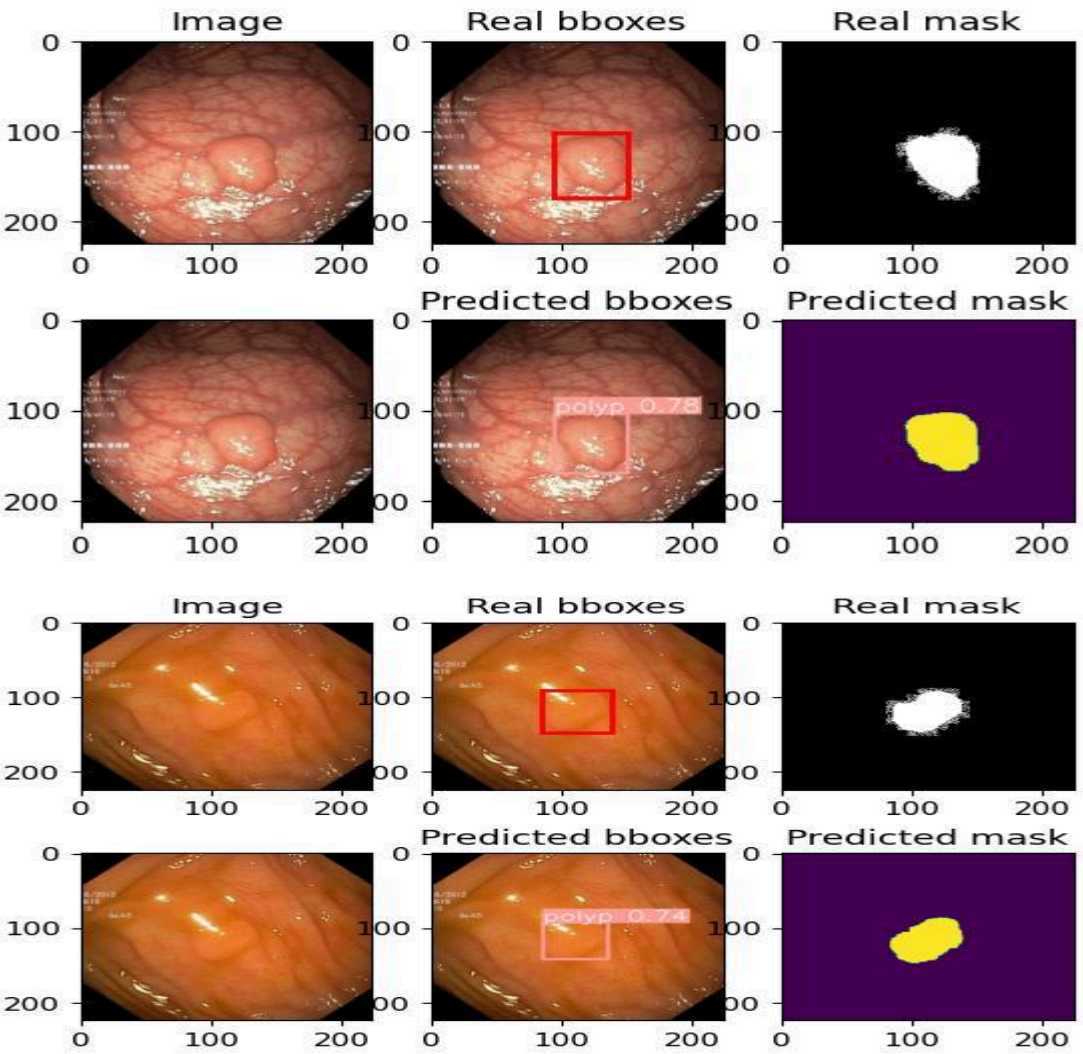


Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
polyp 0.78
Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
polyp 0.74

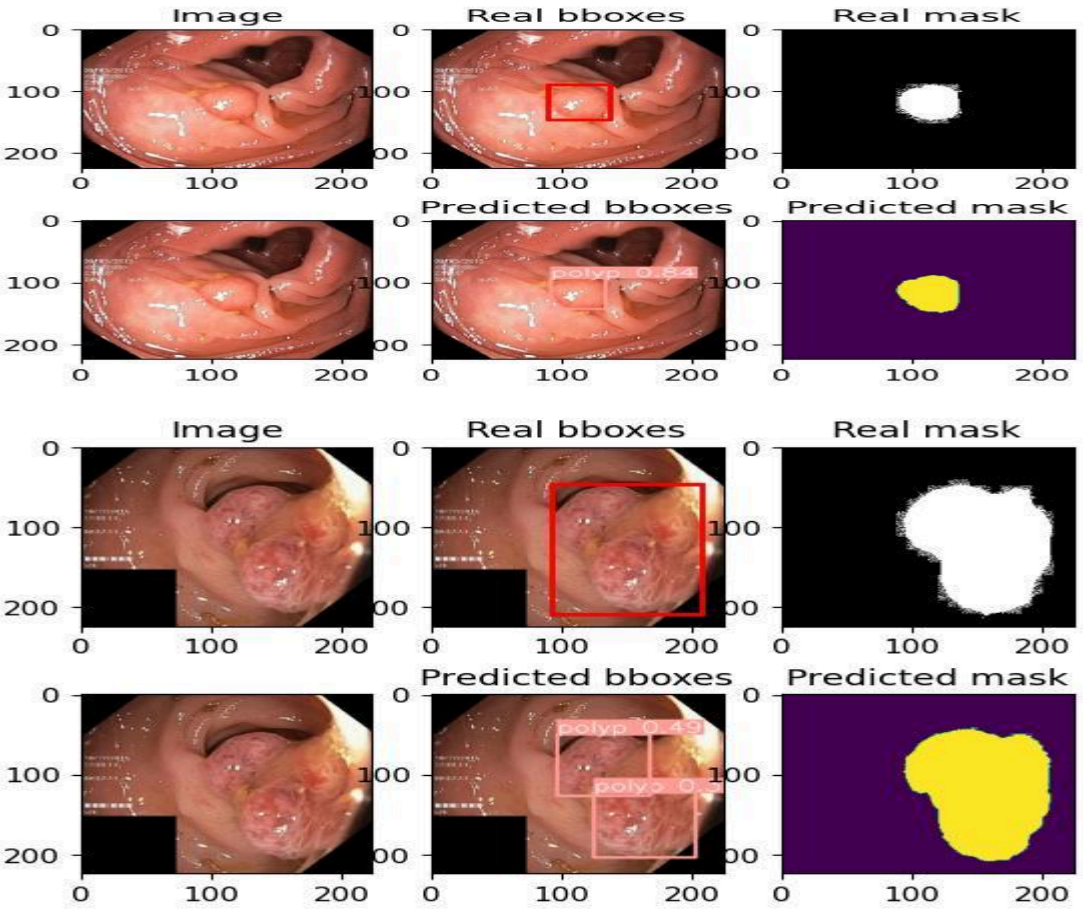


Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
polyp 0.84
Image
Real bboxes
Real mask
Predicted bboxes
Predicted mask
polyp 0.49